CentOS 7.3 MySQL5.7 分区表测试

测试人：马飞

日期：2018-7-27

一、测试环境

|  |  |
| --- | --- |
| 操作系统 | CentOS Linux release 7.3.1611 (Core) |
| 数据库 | 5.7.20-log MySQL Community Server (GPL) |

二、检查是否支持分区

|  |
| --- |
| mysql> SHOW PLUGINS;  ...  | ARCHIVE | ACTIVE | STORAGE ENGINE | NULL | GPL |  | partition | ACTIVE | STORAGE ENGINE | NULL | GPL |  +----------------------------+----------+--------------------+---------+---------+  42 rows in set (0.12 sec)  或者通过查询 INFORMATION\_SCHEMA.PLUGINS 表  mysql> SELECT PLUGIN\_NAME as Name, PLUGIN\_VERSION as Version, PLUGIN\_STATUS as Status  FROM INFORMATION\_SCHEMA.PLUGINS  WHERE PLUGIN\_TYPE='STORAGE ENGINE'; |

说明：由Oracle提供的MySQL 5.6社区版二进制是提供了分区功能支持的。 如果你是源码安装，在cmake时要指定 -DWITH\_PARTITION\_STORAGE\_ENGINE 。如果你想disable分区支持，你可以在启动MySQL时附加–skip-partition选项 MERGE, CSV, or FEDERATED不支持创建分区表。

三、分区表测试

## 3.1 hash 分区

|  |
| --- |
| CREATE TABLE ti (id INT, amount DECIMAL(7,2), tr\_date DATE)  ENGINE=INNODB  PARTITION BY HASH( MONTH(tr\_date) )  PARTITIONS 6; |

注意：ENGINE子句要用在partitioning options前。无论哪种MySQL分区类型,要么分区表上没有主键/唯一键，要么分区表的主键/唯一键都必须包含分区键，也就是说不能使用主键/唯一键字段之外的其他字段分区。下表不能分区，因为唯一键中没有包含分区键id列，即便该列为主键：

## 3.2 range分区

|  |
| --- |
| CREATE TABLE tnp (  id INT NOT NULL AUTO\_INCREMENT,  ref BIGINT NOT NULL,  name VARCHAR(255),  PRIMARY KEY pk (id),  UNIQUE KEY uk (name)  )  partition by range (id) (  partition p0 values less than(10),  partition p1 values less than(20),  partition p2 values less than(30)  );  ERROR 1503 (HY000): A UNIQUE INDEX must include all columns in the table's partitioning function  CREATE TABLE tnp (  id INT NOT NULL AUTO\_INCREMENT,  ref BIGINT NOT NULL,  name VARCHAR(255),  PRIMARY KEY pk (id),  UNIQUE KEY uk (id,name)  )  partition by range (id) (  partition p0 values less than(10),  partition p1 values less than(20),  partition p2 values less than(30)  );  Query OK, 0 rows affected (0.01 sec) |

## 3.3 分区查询

|  |
| --- |
| CREATE TABLE tnp (  id INT NOT NULL AUTO\_INCREMENT,  name VARCHAR(255),  PRIMARY KEY pk (id)  )  partition by range (id) (  partition p0 values less than(10),  partition p1 values less than(20),  partition p2 values less than(30)  );  mysql> insert into tnp values(1,'fan'),(11,'bo'),(21,'shi');  mysql> select \* from tnp;  +----+------+  | id | name |  +----+------+  | 1 | fan |  | 11 | bo |  | 21 | shi |  +----+------+  3 rows in set (0.00 sec)  mysql> select \* from tnp partition(p0,p2);  +----+------+  | id | name |  +----+------+  | 1 | fan |  | 21 | shi |  +----+------+  2 rows in set (0.00 sec) |

说明：MySQL支持只查询某个分区的数据

( DELETE, INSERT, REPLACE, UPDATE, and LOAD DATA, LOAD XML 也支持Partition selection)

## 3.4 分区类型

**RANGE分区**

range分区的表是利用取值范围将数据分成分区，区间要连续并且不能互相重叠，

使用values less than操作符进行分区定义

**LIST分区**

LIST分区是建立离散的值列表告诉数据库特定的值属于哪个分区，LIST分区在很多方面类似于RANGE分区，区别在LIST分区是从属于一个枚举列表的值得集合，RANGE分区是从属于一个连续区间值得集合。

LIST分区通过使用PARTITION BY LIST(expr)子句来实现,expr是某列值或一个局域某列值返回一个整数值得表达式，然后通过VALUES IN(value\_list)的方式来定义分区，其中value\_list是一个逗号分隔的整数列表。

与RANGE分区不同，LIST分区不必声明任何特定顺序

**HASH分区**

HASH分区主要用来分散热点读，确保数据在预先确定个数的分区中尽可能平均分布。对一个表执行HASH分区时，MySQL会对分区键应用一个散列函数，以此确定数据应当放在N个分区中的哪个分区

MySQL支持两种HASH分区，常规HASH分区和线性HASH分区(LINEAR HASH);常规HASH使用的是取模算法，线性HASH分区使用的是一个线性的2的幂的运算法则

**KEY分区**

按照Key进行分区非常类似于按照HASH进行分区，只不过HASH分区允许使用用户自定义的表达式，

而Key分区不允许使用用户自定义的表达式，需要使用MySQL提供的HASH函数

数据库分区的一个非常常见的用途是按日期分离数据，一些数据库系统支持明确的日期分区，

然而Mysql5.7并不支持(明确的)。然后使用DATE, TIME, or DATETIME 列来分区并不是什么难事儿，甚至基于表达式利用这样的列来创建分区。

当通过partition by KEY或LINEAR KEY的方式分区，你可以直接使用 DATE, TIME, or DATETIME column列作为分区键而不用对它们进行任何处理。

|  |
| --- |
| CREATE TABLE members (  firstname VARCHAR(25) NOT NULL,  lastname VARCHAR(25) NOT NULL,  username VARCHAR(16) NOT NULL,  email VARCHAR(35),  joined DATE NOT NULL  )  PARTITION BY KEY(joined)  PARTITIONS 6; |

创建一个存储过程插入一些数据

|  |
| --- |
| delimiter $$  drop procedure if exists pr\_insertdate$$  create procedure pr\_insertdate(in begindate date,in enddate date)  begin  while begindate<enddate  do  insert into members values('fan','boshi','duyalan',null,begindate);  set begindate = date\_add(begindate,interval 1 day);  end while;  end$$  delimiter ; |

调用存储过程插入数据

|  |
| --- |
| mysql> call pr\_insertdate('2015-01-01','2016-08-08');  Query OK, 1 row affected (0.40 sec)  mysql> select count(\*) from members;  +----------+  | count(\*) |  +----------+  | 585 |  +----------+  1 row in set (0.00 sec) |

查看数据分布

|  |
| --- |
| mysql> select  -> partition\_name part,  -> partition\_expression expr,  -> partition\_description descr,  -> table\_rows  -> from information\_schema.partitions where  -> table\_schema = schema()  -> and table\_name='members';  +------+----------+-------+------------+  | part | expr | descr | table\_rows |  +------+----------+-------+------------+  | p0 | `joined` | NULL | 177 |  | p1 | `joined` | NULL | 0 |  | p2 | `joined` | NULL | 208 |  | p3 | `joined` | NULL | 0 |  | p4 | `joined` | NULL | 200 |  | p5 | `joined` | NULL | 0 |  +------+----------+-------+------------+  6 rows in set (0.12 sec) |

说明：然而根据我的实验这种分区貌似没什么卵子用，p1 p3 p5都用不上。这种分区方法也不能很好地利用分区裁剪特性。

## 3.5 日期范围、列表分区

|  |
| --- |
| CREATE TABLE members\_year (  firstname VARCHAR(25) NOT NULL,  lastname VARCHAR(25) NOT NULL,  username VARCHAR(16) NOT NULL,  email VARCHAR(35),  joined DATE NOT NULL  )  PARTITION BY RANGE( YEAR(joined) ) (  PARTITION p0 VALUES LESS THAN (1960),  PARTITION p1 VALUES LESS THAN (1970),  PARTITION p2 VALUES LESS THAN (1980),  PARTITION p3 VALUES LESS THAN (1990),  PARTITION p4 VALUES LESS THAN MAXVALUE  ); |

注意：普通的range和list分区还是要被分区键列转换成整型

创建一个可以传入表名的过程，用到了prepare statement，因为表名无法作为参数传递

|  |
| --- |
| delimiter $$  drop procedure if exists pr\_insertdate\_1$$  create procedure pr\_insertdate\_1(in begindate date,in enddate date,in tabname varchar(40))  begin  while begindate<enddate  do  set @s=concat\_ws(' ','insert into',tabname,'values(''fan'',''boshi'',''duyalan'',null,''',begindate,''')');  prepare stmt from @s;  execute stmt;  drop prepare stmt;  set begindate = date\_add(begindate,interval 1 day);  end while;  end$$  delimiter ; |

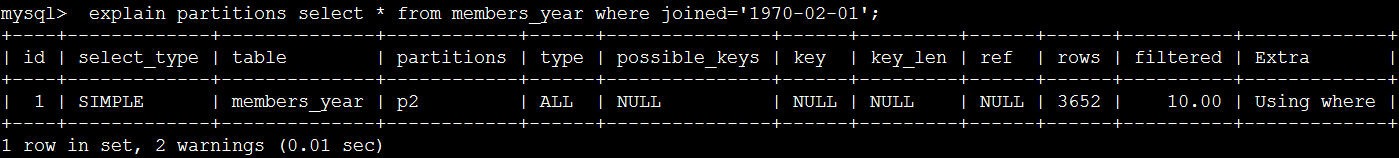
调用存储过程插入数据

|  |
| --- |
| mysql> call pr\_insertdate\_1('1960-01-01','1990-12-31','members\_year');  Query OK, 0 rows affected (6.87 sec)  mysql> select count(\*) from members\_year;  +----------+  | count(\*) |  +----------+  | 11322 |  +----------+  1 row in set (0.01 sec) |

查看数据分布

|  |
| --- |
| mysql> select  -> partition\_name part,  -> partition\_expression expr,  -> partition\_description descr,  -> table\_rows  -> from information\_schema.partitions where  -> table\_schema = schema()  -> and table\_name='members\_year';  +------+---------------+----------+------------+  | part | expr | descr | table\_rows |  +------+---------------+----------+------------+  | p0 | YEAR(joined) | 1960 | 0 |  | p1 | YEAR(joined) | 1970 | 3653 |  | p2 | YEAR(joined) | 1980 | 3652 |  | p3 | YEAR(joined) | 1990 | 3653 |  | p4 | YEAR(joined) | MAXVALUE | 364 |  +------+---------------+----------+------------+  5 rows in set (0.01 sec) |

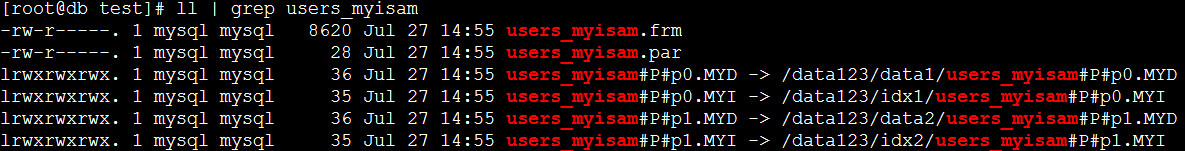
查看是否可以使用到分区裁剪特性



## 3.6 分区表指定目录

对于myisam分区表可以指定DATA Directory、index directory目录：

|  |
| --- |
| chown -R mysql:mysql \*  mkdir -p /data123/data1  mkdir -p /data123/data2  mkdir -p /data123/idx1  mkdir -p /data123/idx2  chown -R mysql:mysql /data123/  CREATE TABLE users\_myisam (  uid INT UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY,  name VARCHAR(30) NOT NULL DEFAULT '',  email VARCHAR(30) NOT NULL DEFAULT ''  ) engine=myisam  PARTITION BY RANGE (uid) (  PARTITION p0 VALUES LESS THAN (10)  DATA DIRECTORY = '/data123/data1'  INDEX DIRECTORY = '/data123/idx1',  PARTITION p1 VALUES LESS THAN (20)  DATA DIRECTORY = '/data123/data2'  INDEX DIRECTORY = '/data123/idx2'  ); |



对于innodb表,只能指定data directory参数：

|  |
| --- |
| CREATE TABLE users\_innodb (  uid INT UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY,  name VARCHAR(30) NOT NULL DEFAULT '',  email VARCHAR(30) NOT NULL DEFAULT ''  )  PARTITION BY RANGE (uid) (  PARTITION p0 VALUES LESS THAN (10)  DATA DIRECTORY = '/data123/data1',  PARTITION p1 VALUES LESS THAN (20)  DATA DIRECTORY = '/data123/data2'  ); |

在指定的路径下会创建 库名文件夹

|  |
| --- |
| [root@db ~]# ll /data123/data1/test  total 96  -rw-r-----. 1 mysql mysql 98304 Jul 31 13:47 users\_innodb#P#p0.ibd  [root@db ~]# ll /data123/data2/test  total 96  -rw-r-----. 1 mysql mysql 98304 Jul 31 13:48 users\_innodb#P#p1.ibd  [root@db ~]# |

在datadir/库名 下包含frm文件和isl文件

|  |
| --- |
| [root@db ~]# ls -l /opt/mysql/data/test | grep innodb  -rw-r-----. 1 mysql mysql 8620 Jul 31 13:47 users\_innodb.frm  -rw-r-----. 1 mysql mysql 41 Jul 31 13:47 users\_innodb#P#p0.isl  -rw-r-----. 1 mysql mysql 41 Jul 31 13:47 users\_innodb#P#p1.isl |

isl文件是一个文本文件,指明了分区ibd文件实际存储路径

|  |
| --- |
| [root@db ~]# file /opt/mysql/data/test/users\_innodb#P#p0.isl  /opt/mysql/data/test/users\_innodb#P#p0.isl: ASCII text, with no line terminators  [root@db ~]# more /opt/mysql/data/test/users\_innodb#P#p0.isl  /data123/data1/test/users\_innodb#P#p0.ibd |

四、分区类型

## 4.1 HASH 分区

按照range分区的表是利用取值范围将数据分成分区，区间要连续并且不能互相重叠，使用values less than操作符进行分区定义。

|  |
| --- |
| CREATE TABLE tnp (  id INT NOT NULL AUTO\_INCREMENT,  name VARCHAR(255),  PRIMARY KEY pk (id)  )  partition by range (id) (  partition p0 values less than(10),  partition p1 values less than(20),  partition p2 values less than(30)  ); |

当插入ID大于30的值是，会出现错误 ，可以在设置分区时使用values less than maxvalue子句。

alter table tnp add partition(partition p3 values less than maxvalue);

MySQL支持在values less than子句中使用表达式，比如，以日期作为range分区的分区列:

|  |
| --- |
| CREATE TABLE emp\_date (  id INT NOT NULL auto\_increment,  name VARCHAR(255),  hiredate date not null default '1970-01-01',  key (id)  )  partition by range (year(hiredate)) (  partition p0 values less than(1995),  partition p1 values less than(2000),  partition p2 values less than(2005)  ); |

MySQL也支持对TIMESTAMP列进行range分区

|  |
| --- |
| CREATE TABLE quarterly\_report\_status (  report\_id INT NOT NULL,  report\_status VARCHAR(20) NOT NULL,  report\_updated TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP  )  PARTITION BY RANGE ( UNIX\_TIMESTAMP(report\_updated) ) (  PARTITION p0 VALUES LESS THAN ( UNIX\_TIMESTAMP('2008-01-01 00:00:00') ),  PARTITION p1 VALUES LESS THAN ( UNIX\_TIMESTAMP('2008-04-01 00:00:00') ),  PARTITION p2 VALUES LESS THAN ( UNIX\_TIMESTAMP('2008-07-01 00:00:00') ),  PARTITION p3 VALUES LESS THAN ( UNIX\_TIMESTAMP('2008-10-01 00:00:00') ),  PARTITION p4 VALUES LESS THAN ( UNIX\_TIMESTAMP('2009-01-01 00:00:00') ),  PARTITION p5 VALUES LESS THAN ( UNIX\_TIMESTAMP('2009-04-01 00:00:00') ),  PARTITION p6 VALUES LESS THAN ( UNIX\_TIMESTAMP('2009-07-01 00:00:00') ),  PARTITION p7 VALUES LESS THAN ( UNIX\_TIMESTAMP('2009-10-01 00:00:00') ),  PARTITION p8 VALUES LESS THAN ( UNIX\_TIMESTAMP('2010-01-01 00:00:00') ),  PARTITION p9 VALUES LESS THAN (MAXVALUE)  ); |

实际是是将TMESTAMP类型通过UNIX\_TIMESTAMP函数转换成INT型:

|  |
| --- |
| mysql> select UNIX\_TIMESTAMP('2008-01-01 00:00:00');  +---------------------------------------+  | UNIX\_TIMESTAMP('2008-01-01 00:00:00') |  +---------------------------------------+  | 1199116800 |  +---------------------------------------+ |

编写过程，动态插入表数据：

|  |
| --- |
| delimiter $$  drop procedure if exists pr\_insertdate\_2$$  create procedure pr\_insertdate\_2(in begindate date,in enddate date,in tabname varchar(40))  begin  set begindate = timestamp(begindate);  while begindate<enddate  do  set @s=concat\_ws(' ','insert into',tabname,'values(1,''true'',''',begindate,''')');  prepare stmt from @s;  execute stmt;  drop prepare stmt;  set begindate = date\_add(begindate,interval 1 day);  end while;  end$$  delimiter ; |

向分区表插入数据：

|  |
| --- |
| mysql> call pr\_insertdate\_2('2007-01-01','2010-12-31','quarterly\_report\_status');  Query OK, 0 rows affected (0.98 sec) |

查询分区表不同分区记录数：

|  |
| --- |
| select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='quarterly\_report\_status';  +------+---------------------------------+------------+------------+  | part | expr | descr | table\_rows |  +------+---------------------------------+------------+------------+  | p0 | UNIX\_TIMESTAMP(report\_updated) | 1199116800 | 365 |  | p1 | UNIX\_TIMESTAMP(report\_updated) | 1206979200 | 91 |  | p2 | UNIX\_TIMESTAMP(report\_updated) | 1214841600 | 91 |  | p3 | UNIX\_TIMESTAMP(report\_updated) | 1222790400 | 92 |  | p4 | UNIX\_TIMESTAMP(report\_updated) | 1230739200 | 92 |  | p5 | UNIX\_TIMESTAMP(report\_updated) | 1238515200 | 90 |  | p6 | UNIX\_TIMESTAMP(report\_updated) | 1246377600 | 91 |  | p7 | UNIX\_TIMESTAMP(report\_updated) | 1254326400 | 92 |  | p8 | UNIX\_TIMESTAMP(report\_updated) | 1262275200 | 92 |  | p9 | UNIX\_TIMESTAMP(report\_updated) | MAXVALUE | 364 |  +------+---------------------------------+------------+------------+  10 rows in set (0.00 sec) |

说明：涉及TIMESTAMP值的任何其他表达式不允许 (See Bug #42849.)

当一个或多个下列条件为真时，范围分区是特别有用的：

（1）需要卸载旧数据

此时你就可以使用 ALTER TABLE employees DROP PARTITION p0;卸载旧数据，效率要比delete高很多

（2）你想用一列包含日期或时间值，或包含从其他系列所产生的价值

You want to use a column containing date or time values, or containing values arising from some other series

（3）where条件列经常为分区列。这个不用多说了，利用分区裁剪特性。

## 4.2 LIST 分区

LIST分区是建立离散的值列表告诉数据库特定的值属于哪个分区，LIST分区在很多方面类似于RANGE分区，区别在LIST分区是从属于一个枚举列表的值得集合，RANGE分区是从属于一个连续区间值得集合。

LIST分区通过使用PARTITION BY LIST(expr)子句来实现,expr是某列值或一个局域某列值返回一个整数值得表达式，然后通过VALUES IN(value\_list)的方式来定义分区，其中value\_list是一个逗号分隔的整数列表。

与RANGE分区不同，LIST分区不必声明任何特定顺序。

下面的例子中我们以下表为基表

|  |
| --- |
| CREATE TABLE employees (  id INT NOT NULL,  fname VARCHAR(30),  lname VARCHAR(30),  hired DATE NOT NULL DEFAULT '1970-01-01',  separated DATE NOT NULL DEFAULT '9999-12-31',  job\_code INT,  store\_id INT  ); |

假设有20家店铺，下表中列出了各个区域包含的店铺ID

Region Store ID Numbers

North 3, 5, 6, 9, 17

East 1, 2, 10, 11, 19, 20

West 4, 12, 13, 14, 18

Central 7, 8, 15, 16

下面按照店铺所属区域进行分区

|  |
| --- |
| CREATE TABLE employees (  id INT NOT NULL,  fname VARCHAR(30),  lname VARCHAR(30),  hired DATE NOT NULL DEFAULT '1970-01-01',  separated DATE NOT NULL DEFAULT '9999-12-31',  job\_code INT,  store\_id INT  )  PARTITION BY LIST(store\_id) (  PARTITION pNorth VALUES IN (3,5,6,9,17),  PARTITION pEast VALUES IN (1,2,10,11,19,20),  PARTITION pWest VALUES IN (4,12,13,14,18),  PARTITION pCentral VALUES IN (7,8,15,16)  ); |

假设西部门店的全部出售给了另一家公司

清空分区：ALTER TABLE employees TRUNCATE PARTITION pWest

删除分区：ALTER TABLE employees DROP PARTITION pWest

LIST分区没有MAXVALUE这种关键字。如果要插入的数据与分区列表不匹配，会引发错误。

|  |
| --- |
| CREATE TABLE h2 (  c1 INT,  c2 INT  )  PARTITION BY LIST(c1) (  PARTITION p0 VALUES IN (1, 4, 7),  PARTITION p1 VALUES IN (2, 5, 8)  );  mysql> INSERT INTO h2 VALUES (3, 5);  ERROR 1526 (HY000): Table has no partition for value 3 |

**说明：**

不过当你使用下面这种MySQL特有的插入语法时，需要注意INSERT INTO h2 VALUES (2, 5), (6, 10), (7, 5), (3, 1), (1, 9); 如果H2表为事务引擎表，如innodb，那么整个插入被视为一个事务，

所以当任何一条记录没有匹配到分区列表时，所有的记录都不被插入 如果H2表为非事务引擎表，如MyISAM，那么引发错误前的记录将被插入，而引发错误的记录包括其后的记录都会插入失败。

|  |
| --- |
| CREATE TABLE h2 (  c1 INT,  c2 INT  ) engine=innodb  PARTITION BY LIST(c1) (  PARTITION p0 VALUES IN (1, 4, 7),  PARTITION p1 VALUES IN (2, 5, 8)  );  mysql> insert into h2 values(1,2),(4,8),(7,8),(3,4),(1,8);  ERROR 1526 (HY000): Table has no partition for value 3  mysql>  mysql> select \* from h2;  Empty set (0.00 sec) |

|  |
| --- |
| CREATE TABLE h2 (  c1 INT,  c2 INT  ) engine=myisam  PARTITION BY LIST(c1) (  PARTITION p0 VALUES IN (1, 4, 7),  PARTITION p1 VALUES IN (2, 5, 8)  );  mysql> insert into h2 values(1,2),(4,8),(7,8),(3,4),(1,8);  ERROR 1526 (HY000): Table has no partition for value 3  mysql> select \* from h2;  +------+------+  | c1 | c2 |  +------+------+  | 1 | 2 |  | 4 | 8 |  | 7 | 8 |  +------+------+  3 rows in set (0.00 sec) |

你可使用IGNORE关键字避免上面的情况，使用IGNORE关键字后，只有引发错误的值将不被插入，其他正常值可以插入，并且不会抛出错误。

|  |
| --- |
| CREATE TABLE h2 (  c1 INT,  c2 INT  ) engine=innodb  PARTITION BY LIST(c1) (  PARTITION p0 VALUES IN (1, 4, 7),  PARTITION p1 VALUES IN (2, 5, 8)  );  mysql> insert IGNORE into h2 values(1,2),(4,8),(7,8),(3,4),(1,8);  mysql> select \* from h2;  +------+------+  | c1 | c2 |  +------+------+  | 1 | 2 |  | 4 | 8 |  | 7 | 8 |  | 1 | 8 |  +------+------+  4 rows in set (0.00 sec) |

注意:在未使用IGNORE前，所有的记录都没有被插入

|  |
| --- |
| CREATE TABLE h2 (  c1 INT,  c2 INT  ) engine=myisam  PARTITION BY LIST(c1) (  PARTITION p0 VALUES IN (1, 4, 7),  PARTITION p1 VALUES IN (2, 5, 8)  );  mysql> insert IGNORE into h2 values(1,2),(4,8),(7,8),(3,4),(1,8);  mysql> select \* from h2;  +------+------+  | c1 | c2 |  +------+------+  | 1 | 2 |  | 4 | 8 |  | 7 | 8 |  | 1 | 8 |  +------+------+  4 rows in set (0.00 sec) |

注意:未使用IGNORE前，最后一条(1,8)也没有被插入

## 4.3 COLUMNS 分区

COLUMN分区是5.5开始引入的分区功能，只有RANGE COLUMN和LIST COLUMN这两种分区；支持整形、日期、字符串；RANGE和LIST的分区方式非常的相似。

（1）针对日期字段的分区就不需要再使用函数进行转换了，例如针对date字段进行分区不需要再使用YEAR()表达式进行转换。

（2）COLUMN分区支持多个字段作为分区键但是不支持表达式作为分区键。

（3）所有数据类型: TINYINT, SMALLINT, MEDIUMINT, INT (INTEGER), and BIGINT。

（4）不支持 Decimal和Float。

（5）日期类型:date和datetime。

（6）字符类型: CHAR, VARCHAR, BINARY, and VARBINARY.

（7）不支持TEXT和BLOB。

### 4.3.1 RANGE COLUMNS 分区

RANGE COLUMNS分区和RANGE分区的几点区别:

（1）RANGE COLUMNS分区键不支持使用表达式，只可以使用列名

（2）RANGE COLUMNS可以支持多列作为分区键进行分区

（3）RANGE COLUMNS分区是基于元组进行比较的

（4）RANGE COLUMNS分区不仅仅限于使用整数型列作为分区键，date和datetime列也可以。

Culumns分区支持非整数分区，这样创建日期分区就不需要通过函数进行转换了。

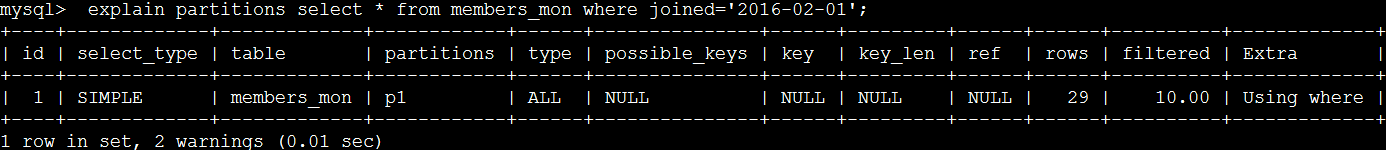
|  |
| --- |
| CREATE TABLE members\_mon (  firstname VARCHAR(25) NOT NULL,  lastname VARCHAR(25) NOT NULL,  username VARCHAR(16) NOT NULL,  email VARCHAR(35),  joined DATE NOT NULL  )  PARTITION BY RANGE COLUMNS(joined) (  PARTITION p0 VALUES LESS THAN ('2016-02-01'),  PARTITION p1 VALUES LESS THAN ('2016-03-01'),  PARTITION p2 VALUES LESS THAN ('2016-04-01'),  PARTITION p3 VALUES LESS THAN ('2016-05-01'),  PARTITION p4 VALUES LESS THAN ('2016-06-01'),  PARTITION p5 VALUES LESS THAN ('2016-07-01'),  PARTITION p6 VALUES LESS THAN ('2016-08-01'),  PARTITION p7 VALUES LESS THAN ('2016-09-01'),  PARTITION p8 VALUES LESS THAN ('2016-10-01'),  PARTITION p9 VALUES LESS THAN ('2016-11-01'),  PARTITION p10 VALUES LESS THAN ('2016-12-01'),  PARTITION p11 VALUES LESS THAN ('2017-01-01'),  PARTITION p12 VALUES LESS THAN MAXVALUE  ); |

向表中插入数据：

|  |
| --- |
| mysql> call pr\_insertdate\_1('2016-01-01','2016-12-31','members\_mon');  Query OK, 0 rows affected (0.23 sec) |

查看分区表数据：

|  |
| --- |
| mysql> select  -> partition\_name part,  -> partition\_expression expr,  -> partition\_description descr,  -> table\_rows  -> from information\_schema.partitions where  -> table\_schema = schema()  -> and table\_name='members\_mon';  +------+----------+--------------+------------+  | part | expr | descr | table\_rows |  +------+----------+--------------+------------+  | p0 | `joined` | '2016-02-01' | 31 |  | p1 | `joined` | '2016-03-01' | 29 |  | p2 | `joined` | '2016-04-01' | 31 |  | p3 | `joined` | '2016-05-01' | 30 |  | p4 | `joined` | '2016-06-01' | 31 |  | p5 | `joined` | '2016-07-01' | 30 |  | p6 | `joined` | '2016-08-01' | 31 |  | p7 | `joined` | '2016-09-01' | 31 |  | p8 | `joined` | '2016-10-01' | 30 |  | p9 | `joined` | '2016-11-01' | 31 |  | p10 | `joined` | '2016-12-01' | 30 |  | p11 | `joined` | '2017-01-01' | 30 |  | p12 | `joined` | MAXVALUE | 0 |  +------+----------+--------------+------------+ |



RANGE Culumns分区还支持多列分区

|  |
| --- |
| create table rc3(  a int,  b int  )  partition by range columns(a,b)(  partition p01 values less than (0,10),  partition p02 values less than (10,10),  partition p03 values less than (10,20),  partition p04 values less than (10,35),  partition p05 values less than (10,MAXVALUE),  partition p06 values less than (MAXVALUE,MAXVALUE)  ); |

注意：RANGE Culumns分区键的比较是基于元组的比较，也就是基于字段组的比较，

这和之前RANGE分区键的比较有些差异，我们写入几条测试数据并观察测试数据的分区情况来看一看。

写入a=1,b=10的记录，实际写入了p02分区，也即是说元组(1,10)<(10,10)

mysql> insert into rc3 values(1,10);

Query OK, 1 row affected (0.00 sec)

|  |
| --- |
| select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='rc3';  +------+---------+-------------------+------------+  | part | expr | descr | table\_rows |  +------+---------+-------------------+------------+  | p01 | `a`,`b` | 0,10 | 0 |  | p02 | `a`,`b` | 10,10 | 1 |  | p03 | `a`,`b` | 10,20 | 0 |  | p04 | `a`,`b` | 10,35 | 0 |  | p05 | `a`,`b` | 10,MAXVALUE | 0 |  | p06 | `a`,`b` | MAXVALUE,MAXVALUE | 0 |  +------+---------+-------------------+------------+  6 rows in set (0.00 sec)  mysql> select (9,34)<(10,10);  +----------------+  | (9,34)<(10,10) |  +----------------+  | 1 |  +----------------+  1 row in set (0.01 sec)  所以insert into rc3 values(9,34);将被插入p02。 |

### 4.3.2 LIST COLUMNS 分区

支持非整型列作为分区键:

（1）字符类型

|  |
| --- |
| CREATE TABLE customers\_1 (  first\_name VARCHAR(25),  last\_name VARCHAR(25),  street\_1 VARCHAR(30),  street\_2 VARCHAR(30),  city VARCHAR(15),  renewal DATE  )  PARTITION BY LIST COLUMNS(city) (  PARTITION pRegion\_1 VALUES IN('Oskarshamn', 'Högsby', 'Mönsterås'),  PARTITION pRegion\_2 VALUES IN('Vimmerby', 'Hultsfred', 'Västervik'),  PARTITION pRegion\_3 VALUES IN('Nässjö', 'Eksjö', 'Vetlanda'),  PARTITION pRegion\_4 VALUES IN('Uppvidinge', 'Alvesta', 'Växjo')  ); |

（2）日期类型

|  |
| --- |
| CREATE TABLE customers\_2 (  first\_name VARCHAR(25),  last\_name VARCHAR(25),  street\_1 VARCHAR(30),  street\_2 VARCHAR(30),  city VARCHAR(15),  renewal DATE  )  PARTITION BY LIST COLUMNS(renewal) (  PARTITION pWeek\_1 VALUES IN('2010-02-01', '2010-02-02', '2010-02-03',  '2010-02-04', '2010-02-05', '2010-02-06', '2010-02-07'),  PARTITION pWeek\_2 VALUES IN('2010-02-08', '2010-02-09', '2010-02-10',  '2010-02-11', '2010-02-12', '2010-02-13', '2010-02-14'),  PARTITION pWeek\_3 VALUES IN('2010-02-15', '2010-02-16', '2010-02-17',  '2010-02-18', '2010-02-19', '2010-02-20', '2010-02-21'),  PARTITION pWeek\_4 VALUES IN('2010-02-22', '2010-02-23', '2010-02-24',  '2010-02-25', '2010-02-26', '2010-02-27', '2010-02-28')  ); |

说明: 不过使用LIST COLUMNS分区进行按时间分区并不太好，还是RANGE COLUMNS好些

|  |
| --- |
| CREATE TABLE customers\_3 (  first\_name VARCHAR(25),  last\_name VARCHAR(25),  street\_1 VARCHAR(30),  street\_2 VARCHAR(30),  city VARCHAR(15),  renewal DATE  )  PARTITION BY RANGE COLUMNS(renewal) (  PARTITION pWeek\_1 VALUES LESS THAN('2010-02-09'),  PARTITION pWeek\_2 VALUES LESS THAN('2010-02-15'),  PARTITION pWeek\_3 VALUES LESS THAN('2010-02-22'),  PARTITION pWeek\_4 VALUES LESS THAN('2010-03-01')  ); |

### 4.3.3 HASH 分区

HASH分区主要用来分散热点读，确保数据在预先确定个数的分区中尽可能平均分布。

对一个表执行HASH分区时，MySQL会对分区键应用一个散列函数，以此确定数据应当放在N个分区中的哪个分区 。

MySQL支持两种HASH分区，常规HASH分区和线性HASH分区(LINEAR HASH)；

常规HASH使用的是取模算法，线性HASH分区使用的是一个线性的2的幂的运算法则（4）RANGE COLUMNS分区不仅仅限于使用整数型列作为分区键，date和datetime列也可以。使用PARTITION BY HASH(expr)创建HASH分区表，expr需要返回一个整数。

（1）常规HASH

下面的例子中创建了一个以store\_id为分区键的hash分区表，如果你没有写partitions子句，那么默认为partitions 1

|  |
| --- |
| CREATE TABLE employees (  id INT NOT NULL,  fname VARCHAR(30),  lname VARCHAR(30),  hired DATE NOT NULL DEFAULT '1970-01-01',  separated DATE NOT NULL DEFAULT '9999-12-31',  job\_code INT,  store\_id INT  )  PARTITION BY HASH(store\_id)  PARTITIONS 4; |

你也可以使用一个返回整型的表达是作为分区键

|  |
| --- |
| CREATE TABLE employees (  id INT NOT NULL,  fname VARCHAR(30),  lname VARCHAR(30),  hired DATE NOT NULL DEFAULT '1970-01-01',  separated DATE NOT NULL DEFAULT '9999-12-31',  job\_code INT,  store\_id INT  )  PARTITION BY HASH( YEAR(hired) )  PARTITIONS 4; |

表达式expr必须返回一个非恒定的，非随机整数值(换句话说，应该是变化，但又是确定的)

同时应当注意的是，该表达式在每次insert update时都会被执行从而决定将数据放入哪个分区，

所以如该表达式的性能低下，会影响整个分区表的性能

MySQL也不推荐使用涉及多列的hash表达式。

常规HASH分区方式看上去挺不错的，通过取模的方式将数据尽可能的平均分布在每个分区中，

让每个分区管理的数据都减少了，提高了查询的效率；课时当我们需要增加分区或者合并分区的时候，

问题就出现了。假设原来是5个常规HASH分区，现在需要新增一个常规HASH分区，

原来的取模算法是MOD(expr,5),根据余数0-4分布在五个分区中，现在新增一个分区，

取模算法编程MOD(expr,6),根据余数0-5分布在6个分区中，原来5个分区中的数据大部分需要通过重新计算重新分区。 常规HASH分区在分区管理上带来的代价太大了，不适合需要灵活变动的需求。

为了降低分区管理上的代价，MySQL提供了线性HASH分区，分区函数是一个线性的2的幂的运算法则。

|  |
| --- |
| create table normal\_hash(  id int  )  partition by hash(id)  partitions 5; |

创建过程向表中插入数据：

|  |
| --- |
| DELIMITER $$  DROP PROCEDURE normal\_insert$$  CREATE PROCEDURE normal\_insert()  BEGIN  DECLARE line INT DEFAULT 0;  WHILE line<1000 DO  INSERT INTO normal\_hash VALUES(line);  SET line=line+1;  END WHILE;  END $$  DELIMITER ; |

生成数据：

|  |
| --- |
| call normal\_insert();  Query OK, 1 row affected (0.38 sec) |

查询分区表数据：

|  |
| --- |
| select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='normal\_hash';  +------+------+-------+------------+  | part | expr | descr | table\_rows |  +------+------+-------+------------+  | p0 | id | NULL | 200 |  | p1 | id | NULL | 200 |  | p2 | id | NULL | 200 |  | p3 | id | NULL | 200 |  | p4 | id | NULL | 200 |  +------+------+-------+------------+  5 rows in set (0.00 sec) |

（2）线性HASH分区

线性HASH分区和常规HASH分区在语法上的唯一区别是在”PARTITION BY”子句中添加LINEAR关键字

|  |
| --- |
| CREATE TABLE employees (  id INT NOT NULL,  fname VARCHAR(30),  lname VARCHAR(30),  hired DATE NOT NULL DEFAULT '1970-01-01',  separated DATE NOT NULL DEFAULT '9999-12-31',  job\_code INT,  store\_id INT  )  PARTITION BY LINEAR HASH( YEAR(hired) )  PARTITIONS 4; |

使用线性HASH是，指定记录保存在哪个分区是可以计算出来的，假设将要保存的记录的分区编号设为N，

num是一个非负整数，表示分割成分区的数量，那么N可以通过以下算法得到。

首先，找到一个大于等于num的2的幂，这个值设为V，V可以通过下面的公式得到

V = POWER(2, CEILING(LOG(2, num)))

例如，刚才创建的employees表预先设定了4个分区，num=4

V = POWER(2, CEILING(LOG(2, 4)))

= POWER(2, CEILING(2))

= POWER(2, 2)

= 4

其次，设置N = F(column\_list) & (v-1)

例如，我们刚才计算出V=4，现在计算stroe\_id=234对于的值

N = F(column\_list) & (v-1)

= 234 & (4-1)

= 2

当 N>=num

设置V=Ceiling(v/2)，这时N=N & (V-1)

对于store\_id=234这条记录，由于N=2<4;所以直接就能够判断这条记录会被存储在第二个分区中

V = POWER(2, CEILING( LOG(2,6) )) = 8

N = YEAR('2003-04-14') & (8 - 1)

= 2003 & 7

= 3

(3 >= 6 is FALSE: record stored in partition #3)

V = 8

N = YEAR('1998-10-19') & (8-1)

= 1998 & 7

= 6

(6 >= 6 is TRUE: additional step required)

N = 6 & CEILING(8 / 2)

= 6 & 3

= 2

(2 >= 6 is FALSE: record stored in partition #2)

有意思的是，当线性HASH分区的个数是2的N次冥时，线性HASH分区的结果和常规HASH分区的结果是一致的。

线性HASH分去的有点事，在分区维护(包括增加、删除、合并、拆分分区)时，

MySQL能够处理的更加迅速;缺点是，对比常规HASH分区(取模)的时候，线性HASH各个分区之间数据的分布不太均衡。

### 4.3.4 Key分区

按照Key进行分区非常类似于按照Hash进行分区，只不过Hash分区允许使用用户自定义的表达式,

而Key分区不允许使用用户自定义的表达式，需要使用MySQL服务器提供的HASH函数;同时Hash分区只支持整数分区，而Key分区支持使用BLOB或Text类型外其他类型的列作为分区键。

我们同样可以使用Partition by key(expr)子句来创建一个key分区表，expr是零个或者多个字段名的列表。与HASH分区不同，创建Key分区表的时候，可以不指定分区键，默认会首先选择使用主键作为分区键。

|  |
| --- |
| CREATE TABLE k1 (  id INT NOT NULL PRIMARY KEY,  name VARCHAR(20)  )  PARTITION BY KEY()  PARTITIONS 2; |

在没有主键的情况，会使用非空唯一键作为分区键

|  |
| --- |
| CREATE TABLE k1 (  id INT NOT NULL,  name VARCHAR(20),  UNIQUE KEY (id)  )  PARTITION BY KEY()  PARTITIONS 2; |

如果作为分区键的唯一键不是非空的(没有指定NOT NULL子句)，依然会报错。

|  |
| --- |
| CREATE TABLE k1 (  id INT ,  name VARCHAR(20),  UNIQUE KEY (id)  )  PARTITION BY KEY()  PARTITIONS 2;  ERROR 1488 (HY000): Field in list of fields for partition function not found in table |

如果即没哟主键，也没有唯一键，就必须要指定分区键了

|  |
| --- |
| CREATE TABLE members (  firstname VARCHAR(25) NOT NULL,  lastname VARCHAR(25) NOT NULL,  username VARCHAR(16) NOT NULL,  email VARCHAR(35),  joined DATE NOT NULL  )  PARTITION BY KEY(joined)  PARTITIONS 6; |

创建过程插入数据：

|  |
| --- |
| delimiter $$  drop procedure if exists pr\_insertdate$$  create procedure pr\_insertdate(in begindate date,in enddate date)  begin  while begindate<enddate  do  insert into members values('fan','boshi','duyalan',null,begindate);  set begindate = date\_add(begindate,interval 1 day);  end while;  end$$  delimiter ; |

插入数据：

|  |
| --- |
| mysql> call pr\_insertdate('2015-01-01','2016-08-08');  Query OK, 1 row affected (0.28 sec)  mysql> select count(\*) from members;  +----------+  | count(\*) |  +----------+  | 585 |  +----------+  1 row in set (0.00 sec) |

五、空值处理

一般情况下，MySQL的分区把NULL当做零值，或者一个最小值进行处理。

## 5.1 RANGE 分区

|  |
| --- |
| create table test\_null(  id int  )  partition by range(id)(  partition p0 values less than (-6),  partition p1 values less than (0),  partition p2 values less than (1),  partition p3 values less than MAXVALUE  );  mysql> insert into test\_null values(null);  Query OK, 1 row affected (0.00 sec)  select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='test\_null';  +------+------+----------+------------+  | part | expr | descr | table\_rows |  +------+------+----------+------------+  | p0 | id | -6 | 0 |  | p1 | id | 0 | 0 |  | p2 | id | 1 | 0 |  | p3 | id | MAXVALUE | 0 |  +------+------+----------+------------+  4 rows in set (0.01 sec) |

当范围分区定义为空时：

|  |
| --- |
| create table test\_null\_1(  id int  )  partition by range(id)(  partition p0 values less than (-6),  partition p1 values less than (0),  partition p2 values less than (1),  partition p2 values less than (null),  partition p3 values less than MAXVALUE  );  ERROR 1566 (HY000): Not allowed to use NULL value in VALUES LESS THAN |

## 5.2 LIST 分区

|  |
| --- |
| create table list\_null(  id int  )  partition by list(id)(  partition p1 values in (0),  partition p2 values in (1)  );  mysql> insert into test\_null values(null);  Query OK, 1 row affected (0.00 sec)  select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='list\_null';  +------+------+-------+------------+  | part | expr | descr | table\_rows |  +------+------+-------+------------+  | p1 | id | 0 | 0 |  | p2 | id | 1 | 0 |  +------+------+-------+------------+  2 rows in set (0.00 sec) |

当分区定义包含NULL时

|  |
| --- |
| create table list\_null\_1(  id int  )  partition by list(id)(  partition p1 values in (0),  partition p2 values in (1),  partition p3 values in (null)  );  mysql> insert into list\_null\_1 values(null);  Query OK, 1 row affected (0.00 sec)  select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='list\_null\_1';  +------+------+-------+------------+  | part | expr | descr | table\_rows |  +------+------+-------+------------+  | p1 | id | 0 | 0 |  | p2 | id | 1 | 0 |  | p3 | id | NULL | 1 |  +------+------+-------+------------+  3 rows in set (0.00 sec) |

## 5.3 HASH、KEY 分区

|  |
| --- |
| create table hash\_null(  id int  )  partition by hash(id)  partitions 2;  mysql> insert into hash\_null values(null);  Query OK, 1 row affected (0.00 sec)  select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='hash\_null';  +------+------+-------+------------+  | part | expr | descr | table\_rows |  +------+------+-------+------------+  | p0 | id | NULL | 1 |  | p1 | id | NULL | 0 |  +------+------+-------+------------+  2 rows in set (0.01 sec) |

六、分区表维护

改变一个表的分区方案只需使用alter table 加 partition\_options 子句就可以了。和创建分区表时的create table语句很像。

## 6.1 创建表

|  |
| --- |
| CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)  PARTITION BY RANGE( YEAR(purchased) ) (  PARTITION p0 VALUES LESS THAN (1990),  PARTITION p1 VALUES LESS THAN (1995),  PARTITION p2 VALUES LESS THAN (2000),  PARTITION p3 VALUES LESS THAN (2005)  ); |

## 6.2 创建插入数据存储过程

|  |
| --- |
| delimiter $$  drop procedure if exists pr\_trb3$$  create procedure pr\_trb3(in begindate date,in enddate date,in tabname varchar(40))  begin  while begindate<enddate  do  set @s=concat\_ws(' ','insert into',tabname,'values(1,''fanboshi'',''',begindate,''')');  prepare stmt from @s;  execute stmt;  drop prepare stmt;  set begindate = date\_add(begindate,interval 1 day);  end while;  end$$  delimiter ; |

## 6.3 调用存储过程插入数据:

|  |
| --- |
| mysql> call pr\_trb3('1985-01-01','2004-12-31','trb3');  Query OK, 0 rows affected (4.31 sec) |

## 6.4 查看数据分布

|  |
| --- |
| select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='trb3';  +------+------------------+-------+------------+  | part | expr | descr | table\_rows |  +------+------------------+-------+------------+  | p0 | YEAR(purchased) | 1990 | 1826 |  | p1 | YEAR(purchased) | 1995 | 1826 |  | p2 | YEAR(purchased) | 2000 | 1826 |  | p3 | YEAR(purchased) | 2005 | 1826 |  +------+------------------+-------+------------+  4 rows in set (0.01 sec) |

## 6.5 改变分区方案

|  |
| --- |
| mysql> ALTER TABLE trb3 PARTITION BY KEY(id) PARTITIONS 4;  Query OK, 7304 rows affected (0.33 sec)  Records: 7304 Duplicates: 0 Warnings: 0 |

## 6.6 查看数据分布

|  |
| --- |
| select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='trb3';  +------+------+-------+------------+  | part | expr | descr | table\_rows |  +------+------+-------+------------+  | p0 | `id` | NULL | 7472 |  | p1 | `id` | NULL | 1826 |  | p2 | `id` | NULL | 1826 |  | p3 | `id` | NULL | 1826 |  +------+------+-------+------------+  4 rows in set (0.00 sec) |

说明：以上结果统计信息有误。

官方：

For partitioned InnoDB tables, the row count given in the TABLE\_ROWS column of

the INFORMATION\_SCHEMA.PARTITIONS table is only an estimated value used in SQL optimization, and is not always exact.

|  |
| --- |
| mysql> select count(0) from trb3 partition (p0);  +----------+  | count(0) |  +----------+  | 7304 |  +----------+  1 row in set (0.01 sec)  mysql> select count(0) from trb3 partition (p1);  +----------+  | count(0) |  +----------+  | 0 |  +----------+  1 row in set (0.00 sec)  mysql> select count(0) from trb3 partition (p2);  +----------+  | count(0) |  +----------+  | 0 |  +----------+  1 row in set (0.00 sec)  mysql> select count(0) from trb3 partition (p3);  +----------+  | count(0) |  +----------+ |

## 6.7 再次改变分区方案

|  |
| --- |
| ALTER TABLE trb3  PARTITION BY RANGE( YEAR(purchased) ) (  PARTITION p0 VALUES LESS THAN (1990),  PARTITION p1 VALUES LESS THAN (1995),  PARTITION p2 VALUES LESS THAN (2000),  PARTITION p3 VALUES LESS THAN (2005)  );  Query OK, 7304 rows affected (0.22 sec)  Records: 7304 Duplicates: 0 Warnings: 0 |

|  |
| --- |
| ALTER TABLE trb3  PARTITION BY RANGE( YEAR(purchased) ) (  PARTITION p0 VALUES LESS THAN (1990),  PARTITION p1 VALUES LESS THAN (1995),  PARTITION p2 VALUES LESS THAN (2000),  PARTITION p3 VALUES LESS THAN (2005)  );  Query OK, 7304 rows affected (0.22 sec)  Records: 7304 Duplicates: 0 Warnings: 0 |

查看数据分布情况：

|  |
| --- |
| select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='trb3';  +------+------------------+-------+------------+  | part | expr | descr | table\_rows |  +------+------------------+-------+------------+  | p0 | YEAR(purchased) | 1990 | 1826 |  | p1 | YEAR(purchased) | 1995 | 1826 |  | p2 | YEAR(purchased) | 2000 | 1826 |  | p3 | YEAR(purchased) | 2005 | 1826 |  +------+------------------+-------+------------+  4 rows in set (0.00 sec) |

## 6.8 改变存储引擎

|  |
| --- |
| drop table trb3;  Query OK, 0 rows affected (0.01 sec)  CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)  PARTITION BY RANGE( YEAR(purchased) ) (  PARTITION p0 VALUES LESS THAN (1990),  PARTITION p1 VALUES LESS THAN (1995),  PARTITION p2 VALUES LESS THAN (2000),  PARTITION p3 VALUES LESS THAN (2005)  );  Query OK, 0 rows affected (0.03 sec) |

插入数据：

|  |
| --- |
| mysql> call pr\_trb3('1985-01-01','2004-12-31','trb3');  Query OK, 0 rows affected (3.86 sec) |

查看数据分布：

|  |
| --- |
| select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='trb3';  +------+------------------+-------+------------+  | part | expr | descr | table\_rows |  +------+------------------+-------+------------+  | p0 | YEAR(purchased) | 1990 | 1826 |  | p1 | YEAR(purchased) | 1995 | 1826 |  | p2 | YEAR(purchased) | 2000 | 1826 |  | p3 | YEAR(purchased) | 2005 | 1826 |  +------+------------------+-------+------------+  4 rows in set (0.00 sec) |

修改存储引擎：

|  |
| --- |
| mysql> alter table trb3 engine=myisam;  Query OK, 7304 rows affected, 1 warning (0.07 sec)  Records: 7304 Duplicates: 0 Warnings: 1 |

查看数据分布：

|  |
| --- |
| select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='trb3';  +------+------------------+-------+------------+  | part | expr | descr | table\_rows |  +------+------------------+-------+------------+  | p0 | YEAR(purchased) | 1990 | 1826 |  | p1 | YEAR(purchased) | 1995 | 1826 |  | p2 | YEAR(purchased) | 2000 | 1826 |  | p3 | YEAR(purchased) | 2005 | 1826 |  +------+------------------+-------+------------+  4 rows in set (0.00 sec) |

## 6.9 查看分区表定义

|  |
| --- |
| mysql> show create table trb3\G  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Table: trb3  Create Table: CREATE TABLE `trb3` (  `id` int(11) DEFAULT NULL,  `name` varchar(50) DEFAULT NULL,  `purchased` date DEFAULT NULL  ) ENGINE=MyISAM DEFAULT CHARSET=utf8  /\*!50100 PARTITION BY RANGE ( YEAR(purchased))  (PARTITION p0 VALUES LESS THAN (1990) ENGINE = MyISAM,  PARTITION p1 VALUES LESS THAN (1995) ENGINE = MyISAM,  PARTITION p2 VALUES LESS THAN (2000) ENGINE = MyISAM,  PARTITION p3 VALUES LESS THAN (2005) ENGINE = MyISAM) \*/  1 row in set, 1 warning (0.00 sec) |

## 6.10 将表由分区表改为非分区表

|  |
| --- |
| mysql> alter table trb3 remove partitioning;  Query OK, 7304 rows affected (0.14 sec)  Records: 7304 Duplicates: 0 Warnings: 0  select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='trb3';  +------+------+-------+------------+  | part | expr | descr | table\_rows |  +------+------+-------+------------+  | NULL | NULL | NULL | 7304 |  +------+------+-------+------------+  1 row in set (0.09 sec)  mysql> show create table trb3\G  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Table: trb3  Create Table: CREATE TABLE `trb3` (  `id` int(11) DEFAULT NULL,  `name` varchar(50) DEFAULT NULL,  `purchased` date DEFAULT NULL  ) ENGINE=MyISAM DEFAULT CHARSET=utf8  1 row in set (0.00 sec) |

## 6.11 将表由分区表改为非分区表

（1）CHECK TABLE, OPTIMIZE TABLE, ANALYZE TABLE, and REPAIR TABLE可以被用于维护分区表。

（2）Rebuilding partitions.相当于将分区中的数据drop掉再插入回来,对于避免磁盘碎片很有效

Example:ALTER TABLE t1 REBUILD PARTITION p0, p1;

（3）Optimizing partitions.如果你的表增加删除了大量数据，或者进行了大量的边长列的更新操作( VARCHAR, BLOB, or TEXT columns)。那么optimize partition将回收未使用的空间，并整理分区数据文件。 Example:ALTER TABLE t1 OPTIMIZE PARTITION p0, p1;

（4）运行OPTIMIZE PARTITION 相当于做了 CHECK PARTITION, ANALYZE PARTITION, and REPAIR PARTITION。

Some MySQL storage engines, including InnoDB, do not support per-partition optimization; in these cases, ALTER TABLE … OPTIMIZE PARTITION rebuilds the entire table. In MySQL 5.6.9 and later, running this statement on such a table causes the entire table to rebuilt and analyzed, and an appropriate warning to be issued. (Bug #11751825, Bug #42822) Use ALTER TABLE … REBUILD PARTITION and ALTER TABLE … ANALYZE PARTITION instead, to avoid this issue.

（5）Analyzing partitions.读取并保存分区的键分布

Example:ALTER TABLE t1 ANALYZE PARTITION p3;

（6）Repairing partitions.修补被破坏的分区

Example:ALTER TABLE t1 REPAIR PARTITION p0,p1;

（7）Checking partitions.可以使用几乎与对非分区表使用CHECK TABLE 相同的方式检查分区。

Example:ALTER TABLE trb3 CHECK PARTITION p1;

这个命令可以告诉你表trb3的分区p1中的数据或索引是否已经被破坏。如果发生了这种情况，使用“ALTER TABLE … REPAIR PARTITION”来修补该分区。以上每个命令都支持将分区换成ALL

（8）mysqlcheck和myisamchk不支持分区表。

（9）你可以使用 ALTER TABLE … TRUNCATE PARTITION. 来删除一个或多个分区中的数据

如:ALTER TABLE … TRUNCATE PARTITION ALL删除所有数据。

（10）ANALYZE, CHECK, OPTIMIZE, REBUILD, REPAIR, and TRUNCATE 操作不支持 subpartitions。

七、范围、列表分区管理

## 7.1 创建表

|  |
| --- |
| drop table trb3;  CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)  PARTITION BY RANGE( YEAR(purchased) ) (  PARTITION p0 VALUES LESS THAN (1990),  PARTITION p1 VALUES LESS THAN (1995),  PARTITION p2 VALUES LESS THAN (2000),  PARTITION p3 VALUES LESS THAN (2005)  ); |

## 7.2插入数据:

|  |
| --- |
| call pr\_trb3('1985-01-01','2004-12-31','trb3');  Query OK, 0 rows affected (3.88 sec) |

## 7.3 查看数据分布

|  |
| --- |
| select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='trb3';  +------+------------------+-------+------------+  | part | expr | descr | table\_rows |  +------+------------------+-------+------------+  | p0 | YEAR(purchased) | 1990 | 1826 |  | p1 | YEAR(purchased) | 1995 | 1826 |  | p2 | YEAR(purchased) | 2000 | 1826 |  | p3 | YEAR(purchased) | 2005 | 1826 |  +------+------------------+-------+------------+  4 rows in set (0.18 sec) |

## 7.4 增加分区

|  |
| --- |
| alter table trb3 add partition (partition p5 values less than(2010));  Query OK, 0 rows affected (0.06 sec)  Records: 0 Duplicates: 0 Warnings: 0 |

## 7.5 合并分区

|  |
| --- |
| alter table trb3 add partition (partition p5 values less than(2010));  select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='trb3';  +------+------------------+-------+------------+  | part | expr | descr | table\_rows |  +------+------------------+-------+------------+  | p0 | YEAR(purchased) | 1990 | 1826 |  | p1 | YEAR(purchased) | 1995 | 1826 |  | p2 | YEAR(purchased) | 2000 | 1826 |  | p5 | YEAR(purchased) | 2010 | 0 |  +------+------------------+-------+------------+  4 rows in set (0.00 sec) |

## 7.6 分裂分区

|  |
| --- |
| mysql> ALTER TABLE trb3 REORGANIZE PARTITION p5 INTO (PARTITION p3 VALUES LESS THAN (2005), PARTITION p4 VALUES LESS THAN (2010));  Query OK, 0 rows affected (0.03 sec)  Records: 0 Duplicates: 0 Warnings: 0  select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='trb3';  +------+------------------+-------+------------+  | part | expr | descr | table\_rows |  +------+------------------+-------+------------+  | p0 | YEAR(purchased) | 1990 | 1826 |  | p1 | YEAR(purchased) | 1995 | 1826 |  | p2 | YEAR(purchased) | 2000 | 1826 |  | p3 | YEAR(purchased) | 2005 | 0 |  | p4 | YEAR(purchased) | 2010 | 0 |  +------+------------------+-------+------------+  5 rows in set (0.00 sec) |

八、HASH KEY分区管理

## 8.1 创建表

|  |
| --- |
| CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)  PARTITION BY hash( YEAR(purchased) )  partitions 12; |

## 8.2 插入数据

|  |
| --- |
| call pr\_trb3('1985-01-01','2004-12-31','trb3'); |

## 8.3 查看数据分布

|  |
| --- |
| mysql> select  -> partition\_name part,  -> partition\_expression expr,  -> partition\_description descr,  -> table\_rows  -> from information\_schema.partitions where  -> table\_schema = schema()  -> and table\_name='trb3';  +------+------------------+-------+------------+  | part | expr | descr | table\_rows |  +------+------------------+-------+------------+  | p0 | YEAR(purchased) | NULL | 731 |  | p1 | YEAR(purchased) | NULL | 365 |  | p2 | YEAR(purchased) | NULL | 365 |  | p3 | YEAR(purchased) | NULL | 365 |  | p4 | YEAR(purchased) | NULL | 366 |  | p5 | YEAR(purchased) | NULL | 730 |  | p6 | YEAR(purchased) | NULL | 730 |  | p7 | YEAR(purchased) | NULL | 730 |  | p8 | YEAR(purchased) | NULL | 732 |  | p9 | YEAR(purchased) | NULL | 730 |  | p10 | YEAR(purchased) | NULL | 730 |  | p11 | YEAR(purchased) | NULL | 730 |  +------+------------------+-------+------------+  12 rows in set (0.00 sec) |

## 8.4 缩减分区

|  |
| --- |
| ALTER TABLE trb3 COALESCE PARTITION 4;  mysql> select  -> partition\_name part,  -> partition\_expression expr,  -> partition\_description descr,  -> table\_rows  -> from information\_schema.partitions where  -> table\_schema = schema()  -> and table\_name='trb3';  +------+------------------+-------+------------+  | part | expr | descr | table\_rows |  +------+------------------+-------+------------+  | p0 | YEAR(purchased) | NULL | 0 |  | p1 | YEAR(purchased) | NULL | 0 |  | p2 | YEAR(purchased) | NULL | 0 |  | p3 | YEAR(purchased) | NULL | 0 |  | p4 | YEAR(purchased) | NULL | 0 |  | p5 | YEAR(purchased) | NULL | 0 |  | p6 | YEAR(purchased) | NULL | 0 |  | p7 | YEAR(purchased) | NULL | 0 |  +------+------------------+-------+------------+  8 rows in set (0.00 sec) |

## 8.5 分区交换

（1）语法

ALTER TABLE pt EXCHANGE PARTITION p WITH TABLE nt

pt是一个分区表，p是pt的分区或子分区，而nt是一个非分区表

（2）限制条件

* 表nt不是分区表
* 表nt不是临时表
* 表pt和nt结构在其他方面是相同的
* 表n没有外键约束，也没有其他表引用它的列为外键
* 表nt的所有行都包含在表p的分区范围内(比如p range分区最大values less than 10，那么表nt不能有大于等于10的值)

（3）权限

除了 ALTER, INSERT, and CREATE 权限外，你还要有DROP权限才能执行ALTER TABLE … EXCHANGE PARTITION.

（4）其他注意事项:

* 执行ALTER TABLE … EXCHANGE PARTITION 不会调用任何在nt表和p表上的触发器
* 在交换表中的任何AUTO\_INCREMENT列会被reset
* IGNORE关键字在执行ALTER TABLE … EXCHANGE PARTITION时会失效
* 在一次ALTER TABLE EXCHANGE PARTITION 中，只能有一个分区和一个非分区表被交换
* 想交换多个，就执行多次ALTER TABLE EXCHANGE PARTITION 任何MySQL支持的分区类型都可以进行交换。

交换实例：

|  |
| --- |
| CREATE TABLE e (  id INT NOT NULL,  fname VARCHAR(30),  lname VARCHAR(30)  )  PARTITION BY RANGE (id) (  PARTITION p0 VALUES LESS THAN (50),  PARTITION p1 VALUES LESS THAN (100),  PARTITION p2 VALUES LESS THAN (150),  PARTITION p3 VALUES LESS THAN (MAXVALUE)  );  INSERT INTO e VALUES  (1669, "Jim", "Smith"),  (337, "Mary", "Jones"),  (16, "Frank", "White"),  (2005, "Linda", "Black"); |

创建一个与e结构一样的非分区表e2：

|  |
| --- |
| mysql> create table e2 like e;  Query OK, 0 rows affected (0.01 sec)  mysql> show create table e2\G  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Table: e2  Create Table: CREATE TABLE `e2` (  `id` int(11) NOT NULL,  `fname` varchar(30) DEFAULT NULL,  `lname` varchar(30) DEFAULT NULL  ) ENGINE=InnoDB DEFAULT CHARSET=utf8  /\*!50100 PARTITION BY RANGE (id)  (PARTITION p0 VALUES LESS THAN (50) ENGINE = InnoDB,  PARTITION p1 VALUES LESS THAN (100) ENGINE = InnoDB,  PARTITION p2 VALUES LESS THAN (150) ENGINE = InnoDB,  PARTITION p3 VALUES LESS THAN MAXVALUE ENGINE = InnoDB) \*/  1 row in set (0.00 sec)  mysql> alter table e2 remove partitioning;  Query OK, 0 rows affected (0.02 sec)  Records: 0 Duplicates: 0 Warnings: 0  mysql> show create table e2\G  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Table: e2  Create Table: CREATE TABLE `e2` (  `id` int(11) NOT NULL,  `fname` varchar(30) DEFAULT NULL,  `lname` varchar(30) DEFAULT NULL  ) ENGINE=InnoDB DEFAULT CHARSET=utf8  1 row in set (0.00 sec) |

查看数据分布：

|  |
| --- |
| select  partition\_name part,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions  where table\_schema = schema()  and table\_name='e';  +------+------+----------+------------+  | part | expr | descr | table\_rows |  +------+------+----------+------------+  | p0 | id | 50 | 1 |  | p1 | id | 100 | 0 |  | p2 | id | 150 | 0 |  | p3 | id | MAXVALUE | 3 |  +------+------+----------+------------+ |

将分区p0与e2表进行交换

|  |
| --- |
| ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2; and table\_name='e';  mysql> select  -> partition\_name part,  -> partition\_expression expr,  -> partition\_description descr,  -> table\_rows  -> from information\_schema.partitions where  -> table\_schema = schema()  -> and table\_name='e';  +------+------+----------+------------+  | part | expr | descr | table\_rows |  +------+------+----------+------------+  | p0 | id | 50 | 0 |  | p1 | id | 100 | 0 |  | p2 | id | 150 | 0 |  | p3 | id | MAXVALUE | 3 |  +------+------+----------+------------+  mysql> select \* from e2;  +----+-------+-------+  | id | fname | lname |  +----+-------+-------+  | 16 | Frank | White |  +----+-------+-------+  1 row in set (0.00 sec) |

重做实验，这次在交换前在表e2中插入一些数据

|  |
| --- |
| mysql> insert into e2 values(16,'FAN','BOSHI');  Query OK, 1 row affected (0.00 sec)  mysql> insert into e2 values(51,'DU','YALAN');  Query OK, 1 row affected (0.00 sec) |

分区交换

|  |
| --- |
| ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;  ERROR 1737 (HY000): Found a row that does not match the partition  报错了，因为51超出了p0的范围,如之前所说，此时使用IGNORE也无济于事。  修改id为49，这样就属于p0的范围了。 |

修改数据

|  |
| --- |
| update e2 set id=49 where id=51;  ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2; |

数据分布：

|  |
| --- |
| mysql> select  -> partition\_name part,  -> partition\_expression expr,  -> partition\_description descr,  -> table\_rows  -> from information\_schema.partitions  -> where table\_schema = schema()  -> and table\_name='e';  +------+------+----------+------------+  | part | expr | descr | table\_rows |  +------+------+----------+------------+  | p0 | id | 50 | 2 |  | p1 | id | 100 | 0 |  | p2 | id | 150 | 0 |  | p3 | id | MAXVALUE | 3 |  +------+------+----------+------------+  4 rows in set (0.00 sec) |

查看交换后表数据：

|  |
| --- |
| e的p0分区中的数据被交换到了e2中\  mysql> select \* from e2;  +----+-------+-------+  | id | fname | lname |  +----+-------+-------+  | 16 | Frank | White |  +----+-------+-------+  1 row in set (0.00 sec) |

## 8.6 子分区交换

准备表

|  |
| --- |
| CREATE TABLE es (  id INT NOT NULL,  fname VARCHAR(30),  lname VARCHAR(30)  )  PARTITION BY RANGE (id)  SUBPARTITION BY KEY (lname)  SUBPARTITIONS 2 (  PARTITION p0 VALUES LESS THAN (50),  PARTITION p1 VALUES LESS THAN (100),  PARTITION p2 VALUES LESS THAN (150),  PARTITION p3 VALUES LESS THAN (MAXVALUE)  );  INSERT INTO es VALUES  (1669, "Jim", "Smith"),  (337, "Mary", "Jones"),  (16, "Frank", "White"),  (2005, "Linda", "Black");  CREATE TABLE es2 LIKE es;  ALTER TABLE es2 REMOVE PARTITIONING; |

尽管我们没有显示的指定每个子分区的名字，我们仍可以通过information\_schema.partitions表获取到子分区的名字。

|  |
| --- |
| select  partition\_name part,  subpartition\_name,  partition\_expression expr,  partition\_description descr,  table\_rows  from information\_schema.partitions where  table\_schema = schema()  and table\_name='es';  +------+-------------------+------+----------+------------+  | part | subpartition\_name | expr | descr | table\_rows |  +------+-------------------+------+----------+------------+  | p0 | p0sp0 | id | 50 | 0 |  | p0 | p0sp1 | id | 50 | 0 |  | p1 | p1sp0 | id | 100 | 0 |  | p1 | p1sp1 | id | 100 | 0 |  | p2 | p2sp0 | id | 150 | 0 |  | p2 | p2sp1 | id | 150 | 0 |  | p3 | p3sp0 | id | MAXVALUE | 3 |  | p3 | p3sp1 | id | MAXVALUE | 0 |  +------+-------------------+------+----------+------------+ |

接下来，开始将p3sp0和es进行交换

|  |
| --- |
| mysql> select \* from es partition(p3sp0);  +------+-------+-------+  | id | fname | lname |  +------+-------+-------+  | 1669 | Jim | Smith |  | 337 | Mary | Jones |  | 2005 | Linda | Black |  +------+-------+-------+  3 rows in set (0.00 sec)  mysql> ALTER TABLE es EXCHANGE PARTITION p3sp0 WITH TABLE es2;  Query OK, 0 rows affected (0.03 sec)  mysql> select \* from es partition(p3sp0);  Empty set (0.00 sec)  mysql> select \* from es2;  +------+-------+-------+  | id | fname | lname |  +------+-------+-------+  | 1669 | Jim | Smith |  | 337 | Mary | Jones |  | 2005 | Linda | Black |  +------+-------+-------+  3 rows in set (0.00 sec) |

如果一个分区表有子分区，那么你只能以子分区为粒度进行交换，而不能直接交换子分区的父分区

|  |
| --- |
| mysql> ALTER TABLE es EXCHANGE PARTITION p3 WITH TABLE es2;  ERROR 1734 (HY000): Subpartitioned table, use subpartition instead of partition |

EXCHANGE PARTITION有着严格的要求，两个将要交换的表的 列名，列的创建顺序，列的数量，以及索引都要严格一致。当然存储引擎也要一致。

|  |
| --- |
| mysql> desc es2;  +-------+-------------+------+-----+---------+-------+  | Field | Type | Null | Key | Default | Extra |  +-------+-------------+------+-----+---------+-------+  | id | int(11) | NO | | NULL | |  | fname | varchar(30) | YES | | NULL | |  | lname | varchar(30) | YES | | NULL | |  +-------+-------------+------+-----+---------+-------+  3 rows in set (0.01 sec)  mysql> create index id\_name on es2(id,fname);  Query OK, 0 rows affected (0.08 sec)  Records: 0 Duplicates: 0 Warnings: 0  mysql> ALTER TABLE es EXCHANGE PARTITION p3sp0 WITH TABLE es2;  ERROR 1736 (HY000): Tables have different definitions |

改变es2的存储引擎

|  |
| --- |
| mysql> drop index id\_name on es2;  Query OK, 0 rows affected (0.01 sec)  Records: 0 Duplicates: 0 Warnings: 0  mysql> alter table es2 engine=myisam;  Query OK, 3 rows affected (0.04 sec)  Records: 3 Duplicates: 0 Warnings: 0  mysql> ALTER TABLE es EXCHANGE PARTITION p3sp0 WITH TABLE es2;  ERROR 1497 (HY000): The mix of handlers in the partitions is not allowed in this version of MySQL |